Results for "((hardware<in>metadata) <and> (firmware<in>metadata))<and> (same <..."

e-mail

Your search matched **16** of **1239820** documents.

A maximum of **100** results are displayed, **25** to a page, sorted by **Relevance** in **Descending** order.

» Search Options

View Session History

New Search

Modify Search

((hardware<in>metadata) <and> (firmware<in>metadata))<and> (same <in>meta >>

Check to search only within this results set

Display Format: Citation Citation & Abstract

» Key

IEEE JNL	IEEE Journal or Magazine
IEE JNL	IEE Journal or Magazine
IEEE CNF	IEEE Conference Proceeding
IEE CNF	IEE Conference Proceeding
IEEE STD	IEEE Standard

Select Article Information

1. **The design and development of a family of personal computers for engineers and scientists**
Nelson, G.E.; Hewlett, W.R.;
Proceedings of the IEEE
Volume 72, Issue 3, March 1984 Page(s):259 - 268
AbstractPlus | Full Text: PDF(1171 KB) IEEE JNL

2. **Simics: A full system simulation platform**
Magnusson, P.S.; Christensson, M.; Eskilson, J.; Forsgren, D.; Hallberg, G.; Hogberg, J.; Larsson, Werner, B.;
Computer
Volume 35, Issue 2, Feb. 2002 Page(s):50 - 58
Digital Object Identifier 10.1109/2.982916
AbstractPlus | References | Full Text: PDF(702 KB) IEEE JNL

3. **A Novel Approach To Real-Time Wave Processing**
Bowers, G.; Kontopidis, G.;
OCEANS
Volume 19, Sep 1987 Page(s):1142 - 1148
AbstractPlus | Full Text: PDF(528 KB) IEEE CNF

4. **Agile methods applied to embedded firmware development**
Greene, B.;
Agile Development Conference, 2004
22-26 June 2004 Page(s):71 - 77
Digital Object Identifier 10.1109/ADEVC.2004.3
AbstractPlus | Full Text: PDF(128 KB) IEEE CNF

5. **Practical experiences in functional simulation. An integrated method from unit to co-simulat**
Schubert, K.-D.;
High-Level Design Validation and Test Workshop, 2002. Seventh IEEE International
27-29 Oct. 2002 Page(s):42
AbstractPlus | Full Text: PDF(224 KB) IEEE CNF

6. **A design tool for fault tolerant systems**
Turconi, G.; Di Perna, E.;
Reliability and Maintainability Symposium, 2000. Proceedi gs. Annual
24-27 Jan. 2000 Page(s):317 - 326

Digital Object Identifier 10.1109/RAMS.2000.816328

AbstractPlus | Full Text: PDF(628 KB) IEEE CNF

7. **Fault coverage design and analysis tools for fault tolerant systems**
Turconi, G.; Di Perna, E.; Marchetti, E.; Valle, R.;
Computer Performance and Dependability Symposium, 1998. IPDS '98. Proceedings. IEEE Interna
7-9 Sept. 1998 Page(s):276
Digital Object Identifier 10.1109/IPDS.1998.707735

AbstractPlus | Full Text: PDF(348 KB) IEEE CNF

8. **A wide swath parametric 3.5 kHz sub-bottom profiling system**
Marino, A.;
OCEANS '95. MTS/IEEE. 'Challenges of Our Changing Global Environment'. Conference Proceedi
Volume 3, 9-12 Oct. 1995 Page(s):2016 - 2020 vol.3
Digital Object Identifier 10.1109/OCEANS.1995.528887

AbstractPlus | Full Text: PDF(340 KB) IEEE CNF

9. **Low cost ANN system based on TDM approach**
Young, D.; Lee Ming Cheng;
Circuits and Systems, 1993., Proceedings of the 36th Midwest Symposium on
16-18 Aug. 1993 Page(s):1185 - 1188 vol.2
Digital Object Identifier 10.1109/MWSCAS.1993.343305

AbstractPlus | Full Text: PDF(296 KB) IEEE CNF

10. **A second generation microprocessor line protection relay**
Peck, D.M.; Engler, F.; De Mesmaeker, I.;
Developments in Power Protection, 1989., Fourth International Conference on
11-13 Apr 1989 Page(s):200 - 204

AbstractPlus | Full Text: PDF(380 KB) IEE CNF

11. **Automatic generation of reprogrammable microcoded controllers within a high-level synthes**
Benmohammed, M.; Rahmoune, A.;
Computers and Digital Techniques, IEE Proceedings-
Volume 145, Issue 3, May 1998 Page(s):155 - 160

AbstractPlus | Full Text: PDF(544 KB) IEE JNL

12. **ASIP micro-code generation from high-level specifications**
Benmohammed, M.; Merniz, S.; Bourahla, M.;
Information and Communication Technologies: From Theory to Applications, 2004. Proceedings. 2
Conference on
19-23 April 2004 Page(s):587 - 588
Digital Object Identifier 10.1109/ICTTA.2004.1307899

AbstractPlus | Full Text: PDF(300 KB) IEEE CNF

13. **802.11a MAC layer: firmware/hardware co-design**
Yeong, J.H.; Rao, X.M.; Shajan, M.R.; Wang, Q.; Lin, J.C.Y.; Qu, X.H.;
Information, Communications and Signal Processing, 2003 and the Fourth Pacific Rim Conference
Proceedings of the 2003 Joint Conference of the Fourth International Conference on
Volume 3, 15-18 Dec. 2003 Page(s):1923 - 1928 vol.3
Digital Object Identifier 10.1109/ICICS.2003.1292802

AbstractPlus | Full Text: PDF(454 KB) IEEE CNF

14. **A comparison of the RTU hardware RTOS with a hardware/software RTOS**
Jaehwan Lee; Mooney, V.J., III; Daleby, A.; Ingstrom, K.; Klevin, T.; Lindh, L.;
Design Automation Conference, 2003. Proceedings of the ASP-DAC 2003. Asia and South Pacific
21-24 Jan. 2003 Page(s):683 - 688

AbstractPlus | Full Text: PDF(887 KB) IEEE CNF

☐ **15. MPEG-2 one-pass variable bit rate control algorithm and its LSI implementation**
Takamura, S.; Kobayashi, N.;
Image Processing, 2001. Proceedings. 2001 International Conference on
Volume 1, 7-10 Oct. 2001 Page(s):942 - 945 vol.1
Digital Object Identifier 10.1109/ICIP.2001.959202
AbstractPlus | Full Text: PDF(296 KB) IEEE CNF

☐ **16. Design and implementation of RSA cryptosystem using multiple DSP chips**
Er, M.H.; Wong, D.J.; Sethu, A.A.; Ngeow, K.S.;
Circuits and Systems, 1991., IEEE International Sympoisum on
11-14 June 1991 Page(s):49 - 52 vol.1
Digital Object Identifier 10.1109/ISCAS.1991.176270
AbstractPlus | Full Text: PDF(256 KB) IEEE CNF

Web Results **1** - **10** of about **122,000** for **hardware firmware hierarchical same numbering**. (**0.21** seconds)

EETimes.com - Making interrupt design **firmware** friendly
However, due to the different reactive speeds of **hardware** and **firmware**, ...
in a **hierarchical** interrupt design, every interrupt source be at the **same** level ...
www.eetimes.com/story/OEG20021017S0048 - 65k - Cached - Similar pages

SDR Forum Primer
The following figure illustrates a high-level **hierarchical** functional model for
... Each subsystem contains **hardware**, **firmware**, an operating system, ...
www.sdrforum.org/sdr_primer.html - 18k - Oct 1, 2005 - Cached - Similar pages

Accelerating system integration by enhancing **hardware, firmware ...**
After a randomly selected **number** of cycles, a new sequence is generated by the
... All **hardware** and **firmware** components can now be covered in the **same** ...
www.research.ibm.com/journal/rd/483/schubert.html - 70k - Cached - Similar pages

[PDF] Accelerating system integration by enhancing **hardware, firmware ...**
File Format: PDF/Adobe Acrobat - View as HTML
multiple steps; for historical reasons, the **numbering** of the ... All **hardware**
and **firmware** components. can now be covered in the **same** environment; ...
www.research.ibm.com/journal/rd/483/schubert.pdf - Similar pages

Making interrupt design **firmware** friendly
A polled status register in a **hardware** design can break a **firmware** program ...
in a **hierarchical** interrupt design, every interrupt source be at the **same** ...
www.us.design-reuse.com/articles/?id=4154&print=yes - 29k - Cached - Similar pages

FirmWorks Open **Firmware** Features
... any **number** of storage devices connected directly to the **hardware** (hard disks
... **Hierarchical** help system for **firmware** functions; stored in a ROM dropin ...
www.firmworks.com/ofwfeatures.htm - 14k - Cached - Similar pages

How to Distinguish Between Different IGX NTM Models [Cisco WAN ...
NTM **Hardware** Model Type. **Firmware** Model Type. Fab/Top Asm **Number** ... Because the
NTM(B) and NTM(C) uses the **same firmware**, you then need to check the card's ...
www.cisco.com/en/US/products/hw/modules/ ps3909/products_tech_note09186a008009419c.shtml - 23k -
Cached - Similar pages

[PDF] AnOptimalApproach:
File Format: PDF/Adobe Acrobat - View as HTML
Emulex **hardware** and **firmware** operate with. any generation of Storport driver,
... Emulex ranked **number** 16 in the Deloitte 2004 Technology Fast 50. ...
www.emulex.com/pdfs/storport.pdf - Similar pages

CESNET Research 2004: Programmable **hardware**
In 2004 we re-organised the team management into a **hierarchical** structure with
... The VHDL workgroup deals with **firmware** design and has 29 active members, ...
www.ces.net/doc/2004/research/proghw.html - 41k - Cached - Similar pages

Embedded.com - eCos Porting Guide
... a **number** of thread priorities, where threads of the **same** priority can be ...
He has developed **hardware**, **firmware**, and device drivers around a variety of ...
www.embedded.com/story/OEG20011220S0059 - 100k - Oct 2, 2005 - Cached - Similar pages

Google Groups results for **hardware firmware hierarchical same numbering**

More Orange-A Guide To Understanding Configuration ... - alt.hackers.malicious - May 16, 2000
Amber-A Guide to Understanding Configuration Management ... - alt.hackers.malicious - May 16, 2000
CURRENT on IBM ThinkPad 600E - muc.lists.netbsd.port.i386 - Jul 06, 2005

Goooooooooogle ▶
Result Page: **1** 2 3 4 5 6 7 8 9 10 **Next**

hardware firmware hierarchical sam Search

Search within results | Language Tools | Search Tips | Dissatisfied? Help us improve

Google Home - Advertising Programs - Business Solutions - Hurricane Resources - About Google

IBM Journal of Research and Development
Volume 48, Number 3/4, 2004
IBM eServer z990
Table of contents: HTML PDF This article: HTML PDF DOI: 10.1147/rd.483.0569

Accelerating system integration by enl hardware, firmware, and co-simulation

by K.-D. Schubert, E. C. McCain, H. Pape, K. Rebmann, P. M. West, and R. Wir

System integration of an IBM eServer™ z990 begins when a z990 book, wh the main processors, memory, and I/O adapters, is installed in a z990 fram Internal Code is "booted" in the service element (SE), and power is turned initial system "bringup," also referred to as post-silicon integration, is com three major steps: initializing the chips, loading embedded code (firmware system, and starting an initial program load (IPL) of an operating system. T processes are serialized, and verification of the majority of the system con cannot begin until they are complete. Therefore, it is important to shorten t time period by improving the quality of the integrated components through comprehensive verification prior to manufacturing. This enhanced coverag on verifying the interaction between the hardware components and firmwa referred to as hardware and software co-simulation). Verification of the act these components first occurs independently and culminates in a pre-silic integration process, or virtual power-on (VPO). This paper focuses primaril hardware subsystem verification of the CLK chip [which is the interface be central electronic complex (CEC) and the service element (SE)] and on enh simulation. It also considers the various environments (collections of hard simulation models, firmware, execution time control code, and test cases t model behavior), with their advantages and disadvantages. Finally, it discu results of the improved comprehensive simulation effort with respect to sy integration for the z990.

Introduction

Success in the server industry is directly related to the features, quality, and dev of a product, and the time it takes to deliver that product to the marketplace. To t implements a very efficient yet comprehensive test strategy to reach a high level Specifically, in the eServer* systems this validation process begins very early in development process with the verification of the **hardware** subsystem and softw

ending with **hardware** and software co-simulation. Subsequently, with the delive engineering **hardware**, the focus is shifted to post-silicon system integration and System test then uses operating systems such as System Assurance Kernel (SA IBM system exerciser, z/OS*, z/VM*, and Linux** to verify the architecture and c functions.

After review of postmortems on server products and analysis of each phase of th development cycle, one basic conclusion can be reached. The time from design **hardware** delivery is determined primarily by the complexity of the design, and th be reduced significantly. However, the time from first engineering **hardware** deli customer availability is driven by testing activities and can be optimized. One wa would be to completely overlap the system integration and system test phases. A approach appears very attractive, but several issues must be considered. Paralle increases the development cost significantly because there is a cost premium or **hardware**. Also, this is difficult because of the serial nature of the system integra stated earlier. For example, one problem involving either **hardware** or software t system integration phase could gate all test activities, thus reducing testing effici unacceptable levels. In the case of a **hardware** problem, it could take weeks or m the problem and fabricate enough **hardware** to move past it. Finally, human reso constraint in conducting many parallel test activities, because more experts are n are available.

Despite these inherent limitations, parallelization is still used for most system inte system test verification work because there are no alternatives at this time. How activity, initial machine load (IML), sometimes referred to as initial microcode loa to the greatest extent by the problems mentioned above, and so cannot be overl because IML is the first step in system integration. Therefore, IBM strategy is to quality of the system components at power-on time in order to reduce the amour needed to reach the parallel phase of system testing.

Another way to optimize would be to understand the nature of certain post-silicon and "move" the verification platform to a less expensive, more efficient and user- platform. If this is done correctly, the negative consequences are minimal. This p the efforts that have been made to significantly increase simulation coverage an function on the cheapest, simplest platform possible (blue areas in **Figure 1**) to n needed for system integration and test (green areas in Figure 1).

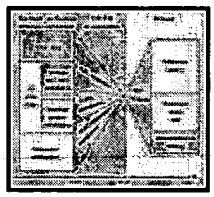
Figure 1

The next section provides an introduction to the system structure to help put sub explanations of the simulation environment in perspective. The following section IML sequence and post-IML simulation environments, including their scope and well as their limitations. Finally, a review of lessons learned and an outlook on fu opportunities for improvement are presented.

System structure

The eServer system consists of **hardware** (HW) and **firmware** (FW) elements. T components are the central electronic complex (CEC), consisting of a shared mu system (SMP) with a memory subsystem, an IBM ThinkPad* used as a service p control the system, the I/O subsystem, and a system control network consisting control chips and cables as shown in **Figure 2**. The power subsystem, which is the scope of the functional simulation activities, is not shown. This subsystem is

with special bringup vehicles (BUVs) prior to the full system power-on (PON).

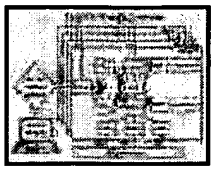
Figure 2

In contrast to previous zSeries* systems, the structure of the CEC comprises fou packaging was necessary to combine up to 48 processor cores in the one syster multibook structure presents new challenges because of the increase in comple system control structure and the CLK chip. The CLK chip is the interface betwee control network [1] and all of the chips in a book. The main tasks of the CLK chip the clocks, shifting the level-sensitive scan design (LSSD) chains of all chips, an fast data path to the processors. To support the multibook structure, additional fu been added to the z990 system. These functions include communication paths a chips in the system and the means of unfencing (logically separating books for s operations) the book-to-book interfaces.

To support all of these functions, the CLK chip has four interfaces, as shown in F establishing the connections to the various targets:

- Service support interface (SSI) to the cage controller (CC).
- Clock service element (CSE) interface to each processor core in a book.
- Clock-to-clock interface (CLK2CLK) for multibook support.
- Serial interface (SIF) to the system control chip (SCC) and storage contro chip in a book.

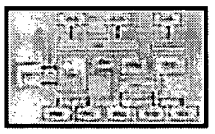
Figure 3

In addition to the **hardware** pieces described above, the system also consists of amount of **firmware**. One part of the **firmware** operates within the CEC and is re providing microprocessor and I/O subsystem control. Another part of the **firmwa** the service processor and is used for functions such as system maintenance, eri and multibook structure support.

IML sequence overview

An important function in a zSeries system is the IML. During IML the **hardware** i and system clocks are started; then millicode and i390 code (henceforth referrec code) are loaded into the CEC. After the CEC code establishes an S/390*-archit state, it is ready to load an operating system and start applications. The IML con operates on the service processor; since this **firmware** is in the critical path of al activities, any problems and delays in debugging it during system integration hav effect on the overall time to market.

IML is a complex process that is broken down into multiple steps; for historical re **numbering** of the steps is not consecutive. **Figure 4** provides an overview of the steps that make up IML. Before any interaction with the service element can be CLK chip must execute a power-on reset (POR) to initialize itself. This is a pure h function which is operated under the control of the run control engine[1] of the CLK engine retains information about the internal chip state and controls the clocks o chips in the system. After the CLK chip has reached the reset state, the connecti service control network is established via the SSI.

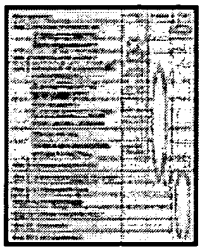
Figure 4

At this point the service element code can access the CLK chip[2] and communica other **hardware** macros (engines) on the chip. To verify that the CLK chip and th the service element are working in lockstep, the sense control engine allows dire the internal control registers of the CLK chip. When a stable connection is establ service element then initializes the other chips by using the shift engine to acces shift latches (chains) of these chips and scan in the appropriate data. After the cl initial state, the first piece of code, the bootstrap code, is loaded into the Level 2 via the serial interface engine on the CLK chip. Thus far, the CLK chip has comn with the service element. With the start of the clocks to the other chips and the e bootstrap code, a new communication path between the processor and the CLK established; this is called the clock service element (CSE) interface. It allows a h between the service element and the processor chips when the special XMessa engine of the CSE on the CLK chip is enabled. With the end of IML step 3, this f communication path is available; all of the megabytes of CEC code that have to from the service element to the CEC in steps 4 and 5 use this path.

In addition to the functions mentioned thus far, functions are available on the CL required only in a multibook system; these have additional verification requireme CLK2CLK engine connects all CLK chips with one another. It is used to synchro of the different books and exchange information between them. Finally, the seria engine and the clock service element interface are reused to fence and unfence cache ring [3].

Of course, the CLK verification does not cover the complete verification of the IM Each of the remaining chips has some logic incorporated to support the IML. The are related primarily to clocking and scanning. Since the functions interface with additional environments[3] are used to verify each of these chips together with the

In addition to the **hardware** functions mentioned so far, the IML sequence relies The two main **firmware** components described in the previous section, the servi and CEC code, are verified separately. After the verification of these elements in environment, they are combined into the CEC simulator, CECSIM [4, 5]; howeve environment contains no **hardware** model, some of the IML code must be bypas

The environments described so far already cover a significant portion of the IML shown in Figure 4, the **hardware** verification focuses on aspects that are used n 2 and 3, while the **firmware** verification covers primarily the service control netw steps 5 to 12. The third environment to be discussed is the one that focuses on t between **hardware** and **firmware**. History has proven that when two separate g the **firmware** and **hardware**, there is a high probability of miscommunication. Th **hardware/firmware** co-simulation (CoSim) was used and enhanced to verify the of design assumptions and interfaces between **hardware** and **firmware**. This ac on the IML steps in which the likelihood of such interface problems is relatively h IML steps 3 and 4, in which **firmware** is performing low-level **hardware** commar valuable overlap among the various activities, the co-simulation also covers part and 5. In the following sections, we describe these activities in more detail and h at which they have been skipped in one environment because they have been v environments.

Hardware verification of the CLK chip

This section focuses on the comprehensive simulation strategy for the CLK chip multilevel verification approach. The scope of verification for each level is determ solving the following optimization problem: Minimize the effort required to verify a properties of a system, including constraints such as start and finish dates.

The first step is to decide how the logic is to be partitioned so that the resultant h tree structure reduces the complexity within each level. The partitions are typical along the boundaries of macros, units, chips, or groups of chips. Each level can a single problem in verification, and a detailed test plan can be constructed by ai characteristics of the design and its interfaces. However, complete coverage of a each level contradicts the goal of minimizing the resources generally—a target o optimization problem mentioned above. This results in decisions such as skippin the test plan on one level in favor of integrated testing one level up in the hierarc

The design of the CLK chip is characterized by a large set of functions with many dependencies among them [6] which are difficult to address on a unit level. Ther decided to skip unit-level verification, leaving the following three verification level chip: macro-level verification, chip-level verification, and multichip-level verificati

Macro-level verification

Since macros are considered the smallest design entity, it is common practice to single designer to implement their design. Each macro provides basic functionali accessible through stimuli on its interfaces. Since there is typically no detailed dc on all of the internals of a macro, the most efficient way to verify it is to have the carry out the macro-level verification. This allows the designer to detect and corr simple problems such as typographic errors on this level.

Chip/multichip-level verification

The chip-level verification is split into two phases, deterministic verification and r verification. Both environments run using the **same** set of tools and share comm components such as drivers and monitors. In chip verification every chip interfac verified using such components. The multibook feature of the z990 eServer resu design in which CLK chips are connected to one another; thus, instead of creatir the CLK2CLK interface, models for two- or four-CLK chips were built to cover thi Using the CLK chip models in this way reduced the workload significantly for the maintaining multiple software behaviorals and models.

The majority of verification tests ran on these models. Before these tests were ru chip initialization file was applied that is common to single-book and multibook cc (i.e., each CLK chip was initialized with the **same** data). The model-build proces chip was extended to generate the initialization file by executing a partial POR s storing the latch values in a file. It should be noted that the **same** initialization file applied in other simulation environments such as **hardware/firmware** co-simula

Deterministic environment

The deterministic verification phase (environment) was implemented before the i verification phase. It is based on a macro language that is built on top of the e pr language.[4] These macros define stimuli on the service support interface (SSI) ar service element (CSE) interface. The SSI is a command-based interface that prc read/write access to the internal registers of the CLK chip. The CSE interface co CLK chip to all processor cores in a book. Like the SSI, it provides read/write ac CLK chip internal registers in the **same** book or via the CLK2CLK interface to the remote books.

All registers accessible via SSI and CSE are located in a set of engines: the shift

dealing with the shifting of internal and external chains; the sense control engine access to the internal clock control registers and run control functions; the XMsg a fast data communication path between the service element and the processing cores; the clock-to-clock engine; and the serial interface engine.

Figure 5 shows the simulation environment for deterministic stimuli. Test cases a sequence of commands, with each command based on the macro language. T then interpreted by the SSI or CSE sequence drivers, which use the lower-level i drivers to execute individual commands. As a result, the interface drivers stimula on the interface. The CLK model is checked for correct behavior, either by comp values to the data returned on the interfaces or by checking model internal regis checking code is implemented in the test case itself.

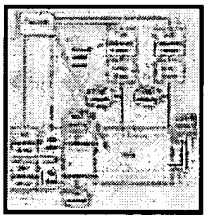
Figure 5

The main purpose of deterministic test cases is to ensure that every logic functio least once. In addition, testing of these logic functions in a sequence that mimics applications such as IML is more complex and must also be verified. Although th of these test cases is a manual, labor-intensive process, they are an important fo the verification of the CLK chip. These deterministic tests are self-checking; they extensive regression package that is extremely valuable in ensuring that design impair the functionality of the chip. The following two examples show how determ are used and how they enhance the whole simulation process.

The first example describes the creation of the CLK chip reset file. This test case of SSI commands to access internal model facilities; it comprises the following st CLK chip is forced into the *Power-On-Reset* state. Second, stimuli from CLK chip are emulated. (Neighbors are the chips that are physically connected to the CLK real system.) Third, the sequence is completed by running the CLK chip into a w *Power-On-Reset-complete* state. At this time, a dump of all latch values is written used in all subsequent environments that include the CLK chip, i.e., the CLK chip itself.

The second example deals with the scan/shift function of the CLK chip. This is a of a simulation which begins with simple test cases and increases its complexity primary contributor to the scan function is the shift engine of the CLK chip and its engine is accessible via the SSI and the CSE interface. The verification begins w checking of the base functions, i.e., access of all registers in the shift engine and internal scan chains such as the CSE chain. It continues by providing test cases and partial shifting, the locking mechanism for concurrent access via SSI and CS partial shift of external chains. Since the shift engine can be accessed from all pl in the system via a remote CLK chip, the next step is to verify the shift mechanis CLK2CLK interface, which includes the sequence for the unfencing of the interfa of the simulation steps described above are the base for the simulation of the pr sparing sequence. In this case all steps which are initiated by millicode later in th must be modeled by a comprehensive simulation sequence.

Random environment

In the deterministic simulation environment illustrated above, scenarios are defin cases. These test cases are generally not fully deterministic, but support parame choose to execute an SSI command on SSI A instead of SSI B). This type of par

improves the coverage, but from a more abstract point of view, these test cases deterministic characteristics.

In the random environment we extend the idea of parameterization significantly ; cover a different state space. The basic idea is to concurrently stimulate the inte CLK chip with randomly selected sequences or just a single command. These se divided into those that leave the state of the logic unchanged or at least in a kno completion (i.e., nondestructive sequences such as a simple read command) an result in an unknown state due to triggered internal activity. The term *unknown s* this context to a state that results from a transition that is too expensive to mode leaves the checking code in an *unknown state.* Thus, these sequences are calle

These destructive sequences are not desirable for the following two reasons:

1. Checking of these scenarios and their side effects is extremely complex a no significant coverage.
2. Some of these sequences are not likely to happen in **hardware** because prevented by code.

However, it is desirable to run permutations of sets of sequences that leave the known state at all times. These sequences are supposed to run concurrently. Fo typical scenario is a shift operation over the SSI interface in combination with so reading or writing to the sense control engine in the local book, and some acces remote books with recoverable error injection. Another scenario would be to hav multiple processors concurrently[6] request the lock to the shift engine.

All sequences applied during a random simulation have variation built in for varia node number, register number, or engine number. The test-case manager maint of all concurrently running sequences, and the resource manager addresses cor between sequences (i.e., determines whether two sequences are allowed to be concurrently). On the basis of this infrastructure, the constraint-driven generatior follows. After a randomly selected number of cycles, a new sequence is generat sequence generator. At this point all degrees of freedom (i.e., engine number) h eliminated. The list of required resources for this sequence is then computed. A solver evaluates whether it is legal to start the new sequence while the current o being executed, according to an algorithm based only on resources. If there is a sequence is rejected; otherwise a new thread is started, and the sequence is ex The execution of sequences is handled with no interaction of the test-case mana Sequences have built-in checking implemented and are driven by SSI and CSE drivers. The sequence drivers utilize the SSI and CSE drivers. This process is si deterministic test-case execution, and some functionality is reused.

Thus far the test-case manager, sequence drivers, and interface drivers have be The other components shown in **Figure 6**, such as the monitors and the referen make up the checking code for the CLK chip[7] environment. Checking is based o commands and is implemented via message passing.

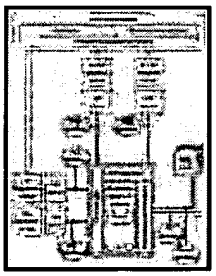

Figure 6

The following example illustrates this checking strategy: for the case "Write com

register #1 of the sense control engine in the local book via processor #4," it beg
CSE driver #4 sending the command over its CSE bus. The corresponding CSE
the data and starts the protocol checking, which will finish on completion of the c
case of an error. While the CSE bus is being monitored, a message is assemble
comprises information on the source processor, target engine, target book, data,
message is sent forward to the CSE interface. The CSE interface checks its con
valid target engine. If this check is successful, it forwards the message to the ref
Within the reference model, the message is associated with the corresponding e
#1 is updated in the reference model, and further actions are triggered if needed
completion of these actions, a message is sent back to the CSE monitor via the
The CSE monitor then compares the predicted result to the response from the **h**
model.

Hardware/firmware co-simulation

Acceleration environment

The terms **hardware/firmware** (HW/FW) co-simulation and VPO imply that this
before real **hardware** is available [7, 8]. To be successful, the VPO concept req
firmware for the initial bringup to be available and simulated by the time at whicl
design is fixed. This concept was enhanced for the z990 system design cycle by
VPO start date earlier, before **hardware** design is fixed, making available more t
silicon" co-simulation test activities. The difficulty in following this strategy is that
time models traditionally change very frequently and are too unstable for meanin
simulation. Therefore, a VPO model snapshot process was invented to closely c
model definition and its associated data files that describe clocking and initializat
the accelerator of choice continues to be the Cadence CoBALT** Ultra system, t
the performance and capacity necessary to handle very large zSeries models. H
a fully configured 12-processor book cannot be built and loaded into the CoBAL
accelerator, tradeoffs were made by deleting noncritical chips for the mainline IM
model most frequently used during the VPO of the latest zSeries 990 system cor
processor chip with two cores, the memory subsystem for one book, one I/O ada
the CLK chip.

To execute **firmware** against that **hardware** model, some way of modeling the s
network is needed. The simplest solution is to strip the service control network d
service element, executing the **firmware** code. The network itself is replaced by
only software layer that connects the output from the laptop to the correct interfa
hardware model on the accelerator. **Figure 7** shows the simulation environment
consists of the laptop running the **firmware**, a workstation to host the special sol
reusing components from other **hardware** simulation activities, and the accelera
the **hardware** model is loaded. The software layer establishes a socket connecti
laptop, and any data traffic from the laptop is translated into commands for a CLI
parallel bus [7, 9] by mimicking the function of the replaced service control netwc
SSI interface that has been verified extensively during the CLK chip verification i
this environment to gain additional simulation speed.

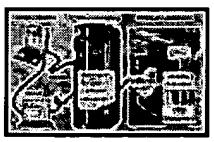
Figure 7

With this setup, the service element code executes as though it were targeting a
z990 system. The major differences between the service element running in sim
connection to a real **hardware** system are twofold. Many of the timeout settings
modified for simulation, since the responses in simulation take significantly longe
all communication between the service element and the power subsystem must
within the **firmware**, emulating real behavior to some extent.

The setup described thus far has been used for most of the co-simulation activiti environment is capable of complete **firmware**-to-**hardware** model interaction, ar constraints that would prevent it from running through the complete IML sequenc to step 12. However, the IML is inherently a sequential process, and simulating t sequence would take at least a month of pure runtime, even on a **hardware** acc capable of executing some 200,000 model cycles per second.

Simulation of IML tasks

For all practical purposes it is not acceptable to have a turnaround time of more of hours, since debugging of problems becomes impossible otherwise. Consequ environment is used only for those parts of the IML process that cannot be verifi by other means. This restricts the co- simulation to IML steps 2 through 5, in whi **hardware/firmware** interaction takes place. In addition, some elements which re simulation cycles can be broken out of this process and verified in a standalone allows us to break the sequential process of IML into tasks that can be attacked avoiding delays when problems are gating progress. This is important because t has to be completed by the time real **hardware** arrives on the test floor.

One of the first tasks to be performed is to prepare a consistent system model st from preceding verification activities such as the CLK chip simulation and CEC s simulation. This shortcut approach does not guarantee that the **same** initial state reached by executing the code on the service element. However, it is close enou machine setup that all subsequent activities behave in the **same** way as on a rea initial state is the starting point for IML step 3 bypassing the requirement to run II the complex array reset process (ABIST reset) usually executed at the beginning Thus, using this shortcut method, the verification of all IML step 3 activities can b immediately, without waiting for debugging of bypassed functions, in effect paral and step 3 debugging.

Since most of the individual steps have already been verified in previous **hardwa firmware** verification activities, this co-simulation environment finds the problem interfaces or in transitions between IML steps. For instance, a class of problems **hardware** initialization is completed and fast communication between the **firmwa hardware** via the XMsg engine must be established. Another hot spot is the eng which describes the initial values for all **hardware** registers. The data is provided **hardware** design team and then processed and to some extent interpreted by th group. During the initial scan operations, this data is shifted into the registers. Si process has been verified during the CLK **hardware** verification, this typically ex problems. However, being able to shift data into registers does not guarantee tha values in the registers are correct. As soon as the chip clocks are started, any er data will likely appear as errors in the **hardware**, which will detect these inconsis the first few hundred cycles. Other areas in which problems are found are the **fir** sequences that control the calibration of the elastic interfaces and also the sectic responsible for initializing the memory and performing the memory self-test. With of the code reset at the end of step 5, this activity is completed.

In parallel with the simulation activity described thus far, initialization shortcuts w However, processes that were bypassed must be verified as well; one of these is reset function. Starting from a state in which the array state is all zeros and only amount of surrounding control logic is set up, array built-in self-test (ABIST) [10] procedures are executed. After this process is completed, the array state is verif yield the **same** state as the shortcut procedures started within IML step 3, provin assumptions that led to the shortcut in the original task.

Another activity is the verification of IML step 3 to step 5 with I/O chips. These I/(initially left out of the model in order to make the model smaller, thereby improvir simulation performance. Also, since much of the IML sequence is independent o any problems with the engineering data of the I/O chips can be corrected while r

progress on the IML without I/O. It is critical to have I/O chips in the model in IML the I/O **hardware** reset is executed and the STI links are initialized.

The environment used for the IML co-simulation can also be used to verify tools later during bringup as debugging aids. The tools are executed on the service el allow different types of accesses to the **hardware**. They typically use scan opera or write certain registers or complete arrays. Tools requiring millicode routines to on their behalf cannot be started before the IML simulation has reached a certain step 4. While tools verification is often viewed as a side activity, its impact on bri integration can be huge. Also, after they have been successfully simulated, som have even been used for debugging the IML co-simulation process itself.

Additional simulation environments

The requirements of multibook simulation are greater than those of the tasks dis previously. The models are by nature at least twice as big, assuming a two-book which is an issue if acceleration capacity and acceleration time are limited. In ad simulation environment is more complex, since the simulation-only software laye two or more independent communication streams. After setting up the environm some simple setup problems in the service element, we terminated that activity, because almost all activities in IML steps 2 to 5 for multibook are identical to tho book IML. In addition, all of the differences have already been verified in the CLK environments. Owing to limited accelerator access, we stopped early in favor of activities described in this paper. In hindsight, this was the correct decision, beca system integration there were no problems that could have been found in this en

The processes described thus far have used an environment in which the compl control network between the service element and the CLK chip has been replace software layer for simulation (Figure 7, option 1). However, the service control ne trivial, since it contains a few chips and executes a significant amount of code as Therefore, in order to increase the simulation coverage of the co-simulation envi discussed thus far, we decided to include at least the code that is executed withi control network. Because the code can be executed in a Linux environment, we component in our environment, as pointed out in option 2 of Figure 7. On one sic system is connected to the service element, and on the other side to the simulati software layer, which had to be slightly modified to support this configuration. As the **same** sequence of IML steps 3 to 5 was executed, focusing only on problem **firmware** code that is executed on the Linux system, since everything else had been verified in the simpler environment.

Results

The first attempts to establish a VPO-like process and to use **hardware-firmwar** simulation for "virtual system integration" or "pre-silicon" were made in 1998. Sin process has been improved continuously in zSeries systems. With the zSeries 9 major breakthrough in VPO simulation has been achieved. Most of the activities front have been successfully co-simulated prior to real PON, many of them for th ever (i.e., IML steps 4 and 5 with I/O chip and bringup tools). The overlap among verification, **firmware** verification, and co-simulation has enabled a very fast brin simulated functions.

During the co-simulation activities after VPO, as shown in **Figure 8**, a significant problems were eliminated from the system. A total of 120 code problems, seven problems, and 91 problems in the simulation environment were found. **Hardwar** found at this point, which is after tape-out (a point in time at which the physical d ready for chip fabrication) cannot be fixed until the next tape-out; however, it is ir state that the problems that would have had a severe impact during system integ circumvented via **firmware** changes, and, more significantly, the circumventions

prior to PON. Thus, by identifying these verification problems and installing **firm** in the system driver, the VPO process has significantly shortened the time requi system bringup. Also, if a severe gating problem was found, weeks would be sa immediately releasing an emergency tape-out.

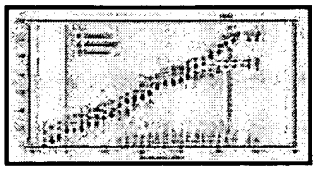
Figure 8

Instrumentation

PICOSIM environment

Thus far, this paper has described simulation activities aimed at minimizing the I time. Since other activities could benefit from similar work, it was logical to exten simulation environment discussed previously in order to achieve additional simul coverage. This initiative has led to the development of the post-IML co-simulatio [11] environment, which encompasses all CEC code (service element code, i39(millicode) and the **hardware** models that have already been used for the co-sim PICOSIM allows the verification of post-IML activities that would normally be exe first time during bringup and system integration.

It is very difficult to set up an environment that is capable of modeling system fur IML is complete. This had been accomplished once, with the simulation effort fo Enterprise System/9000*, but since the shift in technology to CMOS, this enviror been dismantled [12]. One way to create such an environment would be to run a the system model; however, as already mentioned, this would take an extremely even on a CoBALT Ultra accelerator system. To overcome this problem, a proce developed by which z/CECSIM [5], a microcode simulator that runs at zSeries s configured to match the model in PICOSIM. To synchronize both environments, four steps must be executed:

- *Execute the IML.*
  The first step is to execute IML in z/CECSIM with a configuration that corr one in the PICOSIM environment. During the IML sequence, millicode an verify the **hardware** configuration by assigning processor units (PUs), sy processors (SAPs), and channel path IDs (CHPIDs), and allocating a sec to allow communication among processors, i390 code, and millicode subs known as the **hardware** system area, or HSA). The target configuration f verification for the z990 was a 1+1, with one PU and one SAP, which was (Master). This configuration would match the one-cycle model used that i PU chip (two cores), one system control chip (SCC), four SCDs (L2 cach memory storage controllers (MSCs), a memory macro, a two-cycle MBA, a CLK chip.
- *Create a snapshot.*
  Second, a snapshot of all microarchitected facilities and the associated d memory is created from z/CECSIM. In this particular application, after IML the programming model data for millicode and i390 code is saved to a file This includes general-purpose registers, the processor recovery **hardwar** millicode registers, and timing facilities. The data that resides in memory I saved to a file.
- *Load the **hardware** model.*
  The model is pre-initialized with the data from the CEC subsystem **hardw** ensuring a post-IML clock running state for the **hardware**-only environme **hardware** model initialization is updated with this z/CECSIM snapshot of ECC and parity information saved in the earlier step.

- *Transfer the memory image into the **hardware** model.*
  By using the Memmove program (an IBM internal tool used in verification to load and manage the storage hierarchy), the large binary file, which co MB of data, can be loaded into the memory macro.

Upon completion of these steps, the clocks are started and the instruction-fetch begins, with each hierarchy of cache requesting data from the level above it until found in memory. The instructions and operand data are fetched and placed in t and instruction execution continues as usual. The millicode and i390 code go to routines until they receive an outside stimulus such as a system restart [13]; the program status word (PSW) is then set up and points to a small ESAME progran system bootstrap routine. For the z990 system, the instrumentation millicode ha as the first test case for this new environment.

Verification of instrumentation

For the z990, instrumentation millicode verification was selected because it had potential for savings by exploiting the PICOSIM environment. Instrumentation is mechanism used to measure the performance of the IBM eServer z990 system. achieved by iterative execution of instruction streams targeted to stress particula functions and to collect its performance characteristics. The collection is done by selected signals and storing them in **hardware** arrays within the processor. Eac arrays are filled, millicode routines are invoked to move the data from the arrays memory. The data is later used for the analysis of important metrics such as CPI instruction), cache misses, and pipeline stalls.

Since instrumentation is for internal use only, it has little dedicated **hardware**. B **hardware** arrays that are used to collect the instrumentation data require signific on the chip, they are shared with another function. Since the arrays are normally debugging trace data, no debugging traces can be obtained while in instrumenta this reason, instrumentation is extremely difficult to debug.

Historically, instrumentation has suffered from a lack of conventional debugging tools, exacerbated by complex code and **hardware** interactions. Therefore, on p projects this function could be tested only by using real **hardware**, because con **firmware** verification environments do not include support for the instrumentatio

The comprehensive PICOSIM environment currently provides the capability to s complex interactions among **hardware**, i390 code, and millicode, since the mod logic design, including the special instrumentation **hardware** as well as the post- checkpoint from z/CECSIM. In addition to being comprehensive, the PICOSIM e like any simulation environment, is dynamic and flexible. It allows **hardware** faci storage locations to be viewed and altered during test-case execution. In contra on real **hardware**, new test cases can be loaded into storage without bringing th down, and local fixes in the millicode can be applied, using z/CECSIM, without g new image.

While debugging of the environment has been performed on a software simulato environment can easily be moved to **hardware** accelerators to gain the required as extensive simulation runs are required. Verification using the PICOSIM envirc uncovered numerous bugs in millicode, i390 code, and the **hardware** for this sy result, the bringup time for the instrumentation function has been reduced from r months on the predecessor system to only two and a half weeks.

Future work

The environments that have been described thus far have pushed the limits of si further out. The approach of moving activities that have traditionally been execut

system integration into simulation and, within simulation, into the smallest possit has proven to be successful. As the complexity of future systems increases, mor be moved into simulation just to maintain the project cycle at current levels. Targ simulation, such as IML and instrumentation, which have a high potential for sav bringup, have already been covered. However, the investment required for this e be directed to address other scenarios with much less effort but sufficient potent

I/O-related operations are certainly a good example of the requirement for future enhancements. Another area that is already under investigation involves bringup types. While a few bringup tools have already been verified, the verification of ac would make a difference during bringup as well. To mention another example, er testing has been under examination for some time now, and may be feasible witl environments.

To free the required resources for addressing the items mentioned above, it is cr improve the handling and efficiency of the environments. This can be achieved t improvements such as more automation and faster turnaround time, by verifying functions in simpler environments, or even by changing the design to minimize s requirements. This would be the ultimate extension of the strategy presented her

Conclusion

In this paper we present a new strategy that bridges **hardware** and **firmware** ve the goal of optimizing system integration. The effort was driven by the need to sa reduce development cost, and enhance product quality. Through analysis of our simulation environments, enhancements were identified and implemented. All **ha firmware** components can now be covered in the **same** environment; to achieve efficiency, certain verification pieces have been moved into the smallest and the expensive environment possible. Only minimal overlap has been retained to ens of the boundaries between simulation environments.

This new strategy has been successfully implemented and executed for the eSe combined effort of the entire development team has resulted in a significant impr regard to system integration time. A reduction of about eight weeks was achieve with the original system integration plan based on data and experience from pre Simulation efforts using PICOSIM have resulted in similar time savings, especial subsequent chip tape-outs are required, because sufficient feedback from perfor measurements is available much earlier, so that these tape-outs can take place greater confidence of reaching customer quality.

IBM's investment in **hardware/firmware** co-simulation is significant, but well con the result of time saving and reduction in engineering **hardware** required for brin substantial portion of the investment was spent on accelerator **hardware** that wil future projects.

References

Footnotes

*Trademark or registered trademark of International Business Machines Corpora
**Trademark or registered trademark of Linus Torvalds or Cadence Design Syste
[1]**Hardware** macros on the chip are referred to as *engines*.
[2]The service element code interfaces with the books via the service control netw CLK chip.
[3]These consist of collections of **hardware** simulation models, **firmware**, executic code, and test cases to stimulate model behavior.
[4]Verification language provided together with the tool SPECMAN from Verisity, I View, CA.

[5]The XMsg engine is a **hardware** interface on the clock chip that connects the c
to the CEC.
[6]The shift engine has a lock mechanism that ensures exclusive access to it by a
[7]Some checking is done by the test-case manager, as was previously explained.

Received September 18, 2003; accepted for publication March 2, 2004; Internet May 27, 2004

Web Results **1** - **10** of about **110,000** for **hardware firmware hierarchical same management**. (**0.24** seconds)

Microsoft Windows XP - Glossary
The Extensible **Firmware** Interface (EFI) defines a new partition style ...
A logical collection of removable media that have the **same management** policies. ...
www.microsoft.com/resources/documentation/ Windows/XP/all/reskit/en-us/gloss_rk_pro.asp - 346k - Cached - Similar pages

Clustering in a SAN Environment
... in the **hardware**, the identification and **management** of devices on the SAN may
... All HBAs in a cluster must be the **same** type and have the **same firmware** ...
www.microsoft.com/ windowsserversystem/storage/clustering.mspx - 25k - Cached - Similar pages

FirmWorks Open **Firmware** Features
The **same** is true for Telnet. We can setup the **firmware** as a Telnet server ...
Hierarchical help system for **firmware** functions; stored in a ROM dropin module ...
www.firmworks.com/ofwfeatures.htm - 14k - Cached - Similar pages

OVERVIEW
The **hierarchical** help system for **firmware** functions is stored in a ROM drop-in module so it ... Modular interrupt **management**. Useful for **hardware** testing. ...
www.firmworks.com/www/featlist.htm - 31k - Cached - Similar pages

Cisco Aironet Driver and **Firmware** FAQ [Cisco Aironet 350 Series ...
Is the **firmware** the **same** on both cards? If not, is there a later version of **firmware** out for the one ... A. The 4800B and the 340 are the **same hardware**. ...
www.cisco.com/en/US/products/hw/wireless/ ps458/products_qanda_item09186a0080128c0f.shtml - 25k - Oct 2, 2005 - Cached - Similar pages

SDR Forum Primer
The following figure illustrates a high-level **hierarchical** functional model for
... Each subsystem contains **hardware**, **firmware**, an operating system, ...
www.sdrforum.org/sdr_primer.html - 18k - Oct 1, 2005 - Cached - Similar pages

[PDF] ESXMIM/ESXMIM-F2 ETHERNET SWITCH MODULE LOCAL **MANAGEMENT** GUIDE
File Format: PDF/Adobe Acrobat - View as HTML
The **hardware**, **firmware**, or software described in this manual is subject to change without notice. ... In-band network **management** passes data along the **same** ...
www.enterasys.com/support/manuals/**hardware**/1099_02.pdf - Similar pages

ZyXEL Network **Management**
The MIS can simultaneously load the **same firmware** to multiple selected ...
The ZyXEL CNM **management** tool can generate detailed analysis reports as well as ...
www.zyxel.com/product/category. php?indexFlagvalue=1082944528 - 54k - Cached - Similar pages

[PDF] Using Embedded Platform **Management** with WBEM/CIM
File Format: PDF/Adobe Acrobat
to support the common **hardware**,. **firmware** and software elements but ... Figure 1-
CIM offers a **hierarchical** approach to **management** that utilizes providers, ...
www.osatechnologies.com/ CIM%20and%20IPMI%20CTR%20reprint.pdf - Similar pages

[PDF] AnOptimalApproach:
File Format: PDF/Adobe Acrobat - View as HTML
Emulex **hardware** and **firmware** operate with. any generation of Storport driver,
... capabilities—queue **management**,. **hierarchical** resets, and improvements in ...
www.emulex.com/pdfs/storport.pdf - Similar pages

Gooooooooogle ▶
Result Page: **1** 2 3 4 5 6 7 8 9 10 **Next**

hardware firmware hierarchical sam Search

Search within results | Language Tools | Search Tips | Dissatisfied? Help us improve

Google Home - Advertising Programs - Business Solutions - Hurricane Resources - About Google